15 Seconds : Creating a Server Component with Visual Basic

Doug Dean
09/30 / 1998

This article is primarily for Active Server Pages (ASP) developers who would like to take their ASP skills a step further. By bundling ASP code into server-side components, developers not only increase Visual Basic functionality and ASP speed, but also find a marvelous way to encapsulate and protect their ASP source code.

I will walk you through the process of developing a very simple server-side ActiveX ASP component. The emphasis will be on the steps it takes to produce a server-side Dynamic Link Library (DLL) file, not on complex ASP scripting or advanced VB code.

Server-Side Components

First off, server-side ActiveX components should not be confused with client-side ActiveX components. A client-side ActiveX component is sent through a network, along with the HTML information, and will only run on Microsoft's Internet Explorer. Client-side ActiveX components compete with Java components and do not work with Netscape browsers.

A server-side ActiveX component runs on the server that sends the HTML information across the Internet. Therefore, all browsers are compatible with server-side ActiveX components. The server-side ActiveX component adds to the functionality of the server, which serves HTML to visiting browsers. Server-side ActiveX components, therefore, require the server to be compatible rather than the browser. Any server with the appropriate ASP API found in Microsoft's Internet Information Server (IIS) will run server-side ActiveX components.

When IIS is called upon to process code within an ASP file, it parses out the ASP command lines surrounded by <% and %> tags (or between <SCRIPT RUNAT=SERVER> and </SCRIPT> tags). It does this in a timely manner by residing in memory and staying ever accessable for HTML services. Here lies the speed advantage over most CGI programs, which have to initiate execution from EXE files every time a request is issued for their services.

What if you could write ASP code that could attach itself to IIS? Well you can! By using Visual Basic 5, and now VB6, you can create the Dynamic Linked Libraries (DLL files) which will run within the same processing space as IIS and be readily available for service when called upon from within an ASP file. These DLL files are typically business objects that are highly tuned for specific types of processing and number crunching. But HTML code can also be constructed within DLL components that will produce anything that ASP is capable of - and more.

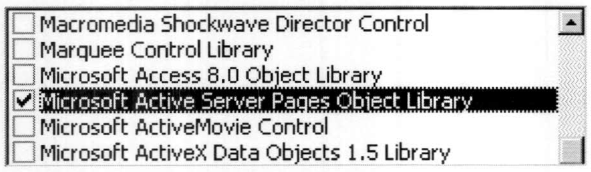
System and Software Requirements

You'll need a 32 bit operating system (Window 95, 98 or NT) running ASP either with IIS or Personal Web Service (PWS). I will be developing an example project on Windows 95, PWS, and Visual Basic 5. It will be necessary to have PWS or IIS installed on your development system. PWS or IIS provides Visual Basic with the references needed to communicate with IIS. If you're already developing ASP files, you probably have PSW or IIS running on your system now. If not, you can download it from http://www.microsoft.com/ie/pws/default.htm?/ie/pws/main.htm.

Setting up Visual Basic

Once you have your ASP enabled operating system ready to go, start Visual Basic and select the ActiveX DLL icon (see figure 1). The ActiveX DLL icon can be found in the 'New Project' window that is typically displayed when Visual Basic is launched. You can also display the 'New Project' window by selecting File/New Project from Visual Basic's menu.

After double-clicking the ActiveX DLL icon , Visual Basic will provide a default project and class for you. We will rename both the Project1 and Class1 to our liking. Before we do that, let's establish a reference to 'Microsoft Active Server Pages Object Library' which is necessary for developing code using ASP. From Visual Basic's menu, select 'Project' and then ''References'. A reference window (see figure 2) will display all the references available on your particular operating system. Scroll down the list until you come to the 'Microsoft Active Server Pages Object Library' reference.


[ ] Macromedia Shockwave Director Control
[ ] Marquee Control Library
[ ] Microsoft Access 8.0 Object Library
[✓] Microsoft Active Server Pages Object Library
[ ] Microsoft ActiveMovie Control
[ ] Microsoft ActiveX Data Objects 1.5 Library

Check the box next to this reference and press the 'OK' button. If you don't find this reference, you'll need to go back to square one and set your operating system up with PWS or IIS.

Name the Project and the Class

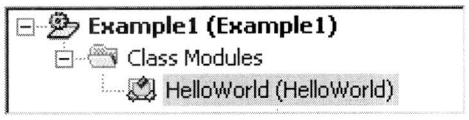
Now let's personalize our project and change the default Project1 and Class1 names that Visual Basic supplied for us. It is important to choose descriptive names because we will be using these very same names to reference our object within an ASP file. Since this is an introductory article on how to create server-side ActiveX components, it seems fitting to write a Hello World program -- with a little ASP twist of course.

We'll call out project 'Example1' and our class 'HelloWorld'. Eventually we'll be writing a class method named 'SayHello'.

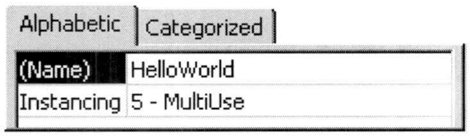
If the Project Explorer (figure 3) is not already displayed, select 'View/Project Explorer' from Visual Basic's menu. You'll also need the Properties window (figure 4) showing, which can be displayed by selecting 'View/Properties Window' from the menu. The Properties window is known to hide down at the bottom of the screen sometimes, so look for it down there and expand its window if you don't see enough of it.

Now click on the Project1 name within the Project Explorer and look at the Properties window. It shows the name of our project to be Project1. Highlight the Project1 text to the right of the (Name) label within the Properties window. Change it to 'Example1'.

Now that we have a name for our project, we will want to name our class from Class1 to HelloWorld. In the Project Explorer window, click on Class1. If you don't see the Class1 name, and only the Class Module's text is showing, click the plus icon within the square to display the class name. Now, down in the Proprieties window, highlight the Class1 text next to the (Name) label and change it to "HelloWorld'


Example1 (Example1)
Class Modules
HelloWorld (HelloWorld)

You'll also notice an 'Instancing' label within the Properties window. Clicking the text to the right of this label produces a drop-down list with various selections. Choose number 5, MultiUse, for our project. Also, select Project/Example1 Properties... and select Apartment Threaded in the drop-down box in the lower right-hand corner of the Project Properties window. This will allow the class we write to generate multiple copies of itself within memory each time a user links to the ASP page that uses our component.



(Name)	HelloWorld
Instancing	5 - MultiUse

How the Project and the Class Names are Used

We now have our project (Example1) and class name (HelloWorld). As mentioned before, we will be using these names to reference our code from an ASP file. This is the same process used for instantiating Microsoft's built-in components. The basic syntax is:

```
Set ObjReference = Server.CreateObject("ProjectName.ClassName")
```

For our example we will use the same syntax and replace the ProjectName and ClassName with Example1 and HelloWorld.

```
Set ObjReference = Server.CreateObject("Example1.HelloWorld")
```

Now by using the ObjReference, which points to the object we will be making, your ASP files can call upon any public Function or Sub you create within your Visual Basic project. We will be writing a Sub named SayHello, so accessing this Sub will be as simple as writing the following with an ASP file:

```
<%
Set ObjReference = Server.CreateObject("Example1.HelloWorld")
ObjReference.SayHello
%>
```

But we've gotten ahead of ourselves. We have to write our Sub for this to work.

Utilizing the ScriptingContext

In order to use ASP methods within the HelloWorld class, you'll have to place a prescribed OnStartPage

Sub within your class code. So write the following Sub within the HelloWorld code window (View/Code from the menu).

```
Public Sub OnStartPage(PassedScriptingContext As ScriptingContext)
    Set MyScriptingContext = PassedScriptingContext
End Sub
```

Now, whenever a user visits an ASP file containing our component, IIS will pass the ScriptingContext to our object for us to use. This ScriptingContext contains all of the Active Server Page methods and properties that are available to the ASP file. In fact, let's assign all of the ASP objects to object variables within this same Sub so they will be available if we need them.

```
Public Sub OnStartPage(PassedScriptingContext As ScriptingContext)
    Set MyScriptingContext = PassedScriptingContext
    Set MyApplication = MyScriptingContext.Application
    Set MyRequest = MyScriptingContext.Request
    Set MyResponse = MyScriptingContext.Response
    Set MyServer = MyScriptingContext.Server
    Set MySession = MyScriptingContext.Session
End Sub
```

So that the Visual Basic compiler won't complain, write the following code declaring these ASP objects above the OnStartPage Sub you just wrote.

```
   Private MyScriptingContext As ScriptingContext
   Private MyApplication As Application
   Private MyRequest As Request
   Private MyResponse As Response
   Private MyServer As Server
   Private MySession As Session
```

Using ASP Objects

Our component object variables can be used in the same way they are used in a typical ASP file. For example, you may have an ASP file in which you use data gathered from an HTML form. If a text input control within this form is named UserName, the ASP file designated within the form's action parameter would have access to the data typed into a text input box. Here's the ASP code that captures the user's input:

```
<%
MyTempVariable = Request.Form("UserName")
MyResponse.Write ("You entered " & MyTempVariable & " as your user name")
%>
```

To do the same thing in our class we would write:

```
MyTempVariable = MyRequest.Form("UserName")
MyResponse.Write("You entered " & MyTempVariable & " as your user name")
```

By using 'MyResponse' in place of 'Response', we can use any of the ASP Response methods. Of course, I made up the MyResponse name. You can substitute any other legal variable name for MyResponse when you define it. You can even match ASP's terminology and name the object variable 'Response'. Your component ASP code would then look identical to ASP file code (except for the tags). The same is true for the other ASP object variables. For instructive purposes, I chose to prefix my object variables with 'My" in order to show the distinction between their 'given' names and my 'chosen' variable names.

One more thing we should do before starting our SayHello Sub is write a Sub that runs when IIS has completed using an instance of our HelloWorld class. Just as the OnStartPage is called whenever our class is referenced by an ASP file, the OnEndPage is called when our object is no longer being used. Here is where we can clean up and set the objects we created to Nothing by using the following Sub.

```
Public Sub OnEndPage()
    Set MyScriptingContext = Nothing
    Set MyApplication = Nothing
    Set MyRequest = Nothing
    Set MyResponse = Nothing
    Set MyServer = Nothing
    Set MySession = Nothing
End Sub
```

The SayHello Method

Our first objective will be simply to print "Hello World" from our component to any browser fortunate enough to visit our ASP page. By using the MyResponse.Write() method within a Public Sub named 'SayHello', we define a method that can be accessed by any ASP page using our Example1 component.

Again, look at the ASP code that will call our Sub (see below). Notice that the object variable (ObjReference), which we set to our project and class names, is also used to call the SayHello Sub we will be writing.

```
<%
Set ObjReference = Server.CreateObject("Example1.HelloWorld")
ObjReference.SayHello
%>
```

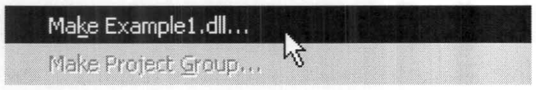
Now the big moment we have been preparing for - writing our SayHello method. Along with the variable definitions, OnStartPage Sub, and OnEndPage Sub which we already wrote, add the following code.

```
Public Sub SayHello()
```

```
    MyResponse.Write ("Hello World")
End Sub
```

That's it for the coding (I warned you it would be simple code!). Now save you program by selecting 'File/Save Project' from the Visual Basic menu. Choose the default name, HelloWorld.cls, provided in the 'Save File As' window. Do the same with the Example1.vbp default shown in the 'Save Project As' window -- which will immediately be displayed after the 'Save As File' window. Then press the Save button.

Although this saves our code, it doesn't create the DLL file we need. If this is the first time you are creating this particular DLL file, simply select 'File/Make Example1.dll…' from the Visual Basic menu (figure 4). Visual Basic will compile the program, notify you of any errors that need to be corrected, and register the Example1.dll on your system.



Making a DLL Component After Using It in an ASP File

If you have gone through the process of making a DLL, and used the DLL within an ASP file, you'll need to shut down your Personal Web Server (PWS) in order to compile the DLL again (please ask Microsoft why). Visual Basic will complain that the DLL file can not be compiled because the component is in use by IIS -- even if you use the server manager's stop option.

I have created two shortcut icons on my desktop which run batch files for stopping and starting my PWS. This allows me to compile DLL components after testing them on my PWS.

Since I'm running PWS, I wrote a simple batch (.bat) file with Notepad. I named it pwsStop.bat and saved it within the same directory where my pws.exe is found (on my system pws.exe lives in the C:\WINDOWS\SYSTEM\INETSRV\PWSSTOP directory). The bat file contains the following command:

```
pws /stop
```

Conversely, I saved a bat file named pwsStart in the same directory with the following command line:

```
pws /start
```

By using Explorer (the file viewer – not the browser), I right-clicked on each bat file and created a shortcut that I dragged onto my desktop. Now, after testing a component within my PWS, I can double click the shortcut to pwsStop and then successfully compile my component again. When I want to test it, I double-click my pwsStart icon.

Writing an ASP File For Our Component

After you correct any compilation errors, and successfully compile the Example1 Project, get your favorite HTML editor out. We will need an ASP file to try out our Example1 component.

Create an ASP file named Example1.asp containing the following code:

```
<HTML>
<HEAD>
           <TITLE>Example 1</TITLE>
</HEAD>

<BODY>

<%
Set ObjReference = Server.CreateObject("Example1.HelloWorld")
ObjReference.SayHello
%>

</BODY>
</HTML>
```

After saving this ASP file, you can view the page with your browser through your PWS, intranet, or Internet. If you try to open this ASP page as a computer file, it won't be processed through PWS or IIS and your component won't work. Since your PWS has its own URL, you can use it to engage PWS and put your component to work. Since I saved my component and ASP file in the 'C:/wwwroot/Example1' directory, I use the following URL:

```
http://190.0.0.1/Example1/Example1.asp
```

Registering Components on Other Systems

If you want to share your components with your friends and neighbors they will have to register the component's DLL file on their own system. They can register any component using the Regsvr32.exe file, typically found in the Windows/System directories on Windows 95 and 98. Have them select Start/Run from the Windows operating system. They will have to type in Regsvr32.exe, a space, and then the full path and the name of the component to register. Since my component was saved in the 'C:/wwwroot/Example1' directory path, I could register my component with the following command line:

```
Regsvr32.exe C:/wwwroot/Example1/Example1.dll
```

Again, Visual Basic will do this for you on your system. So you'll rarely need to use Regsvr32.exe yourself.

Applications for Components

Obviously, we have come a long way just to display two words that would have taken two seconds using plain HTML. Since this is an article about the procedures for creating any ActiveX server-side component, I wanted to remove extraneous complexity so you could clearly see the process before your creative juices began to flow. If you're like me, once I know how to apply a certain technology, my mind doesn't stop until have a few applications all thought out.

One component can produce a single page displaying simple HTML text, or it can display computed data originating from database tables specifically designed for individual users or groups. An entire interactive site, with multiple related pages, can be contained in one component. Shopping cart systems, a series of reusable input forms with built in data validation, office related information management sites, and many other application can be encapsulated and saved within a single DLL file. The class libraries you build can be saved and reused in other components. As an added bonus, distributing compiling source code in the form of a DLL component will keep your code manageable and protected.

Expanding The SayHello Method

Let's take our simple SayHello Sub and expand it a little to get a flavor of what can to be done with a component like this.

When you write HTML code that allows your visitors to hyperlink to another HTML page, you can also add a little extra information to the end of the hyperlink URL line. This is an excellent technique for keeping track of multiple pages within a single component. Let's make our 'Hello World' text a hyperlink with an attached query string. The hyperlink HREF will point to the same file it is written in (Example1.asp). We can then write code to keep track of how many times a user has sequentially pressed our Hello World hyperlink -- all without using an ASP Session variable.

A Note on ASP Session Variables

Much of the ASP code I see uses Session variables to accomplish the same effect that query strings can achieve. I try to avoid Session variables for several reasons. If your ASP page uses a Session variable, a cookie must be sent and retrieved from your server in order to keep track of the user. Many Session variables stay alive 20 minutes after that user has surfed off to other sites. Although most browsers have the capacity to read and write the cookie necessary for this, many users turn the cookie feature off. Session variables will not work when this happens.

Along with producing a little more web traffic, Session variables also use more server system resources than query strings (and even a little overhead can add up fast on hot sites). I also like using a query string for keeping track of users' whereabouts because it helps me manage my pages in more of a modular way -- keeping down global variable use. Session variables are invaluable and irreplaceable at times. But they are frequently overused and unnecessary much of the time.

The Expanded SayHello Code

Expanding the SayHello Sub is not hard and demonstrates some of the fundamental principles for sending Dynamic HTML code back to the server via a component.

Here are the changes to make:

```
Public Sub SayHello()
```

```
    '----- Declare the variables used in this Sub
    Dim strVariable As String
    Dim intCount As Integer

    '----- Get the Query String, if any, and increase it by one
    intCount = MyRequest.QueryString("HelloCount")
    intCount = intCount + 1

    '----- Constuct the HTML code to send back to the browser
    strVariable = "<A HREF=""Example1.asp?HelloCount="
    strVariable = strVariable & intCount & """>Hello World</A>"
    strVariable = strVariable & "<BR><BR>You have pressed ""Hello World"" "
    strVariable = strVariable & intCount - 1 & " times in a row."

    '----- Send the constructed HTML code
    MyResponse.Write (strVariable)

End Sub
```

The first thing you'll notice is the use of the declared variables 'strVariable' and 'intCount'. The 'strVariable' variable is used to collect our HTML string before we send it back to the server in one lump sum. The 'intCount' variable stores the passed query string named 'HelloCount' attached to the end of the URL linked back to our ASP page.

Think of the MyRequest.QueryString("HelloCount") as the value assigned to the query string name that's attached to the end of the URL. A hyperlink tag like <A HREF="Example1.asp?HelloCount=3"> would produce a query string of HelloCount=3. So MyRequest.QueryString("HelloCount") is equivalent to '3'.

Our code placed the query string value in the intCount variable and then increased it by one. The same could be accomplished more concisely with the following code:

```
IncCount = MyRequest.QueryString("HelloCount") - 1
```

or even directly included in the strVariable without the use of the intCount variable.

```
strVariable = strVariable & MyRequest.QueryString("HelloCount") - 1 & " times in a row."
```

Also notice the line that produces the hyperlink tag:

```
strVariable = "<A HREF ""Example1.asp?HelloCount=" & intCount & """>Hello World</A>"
```

This code line not only demonstrates the way to concatenate string and integer variables ("string" & integer & "string"), but also how to escape the quote (") character ("”this line in quotes”"). The double quotes are needed for the file name and query string attachment. When an HTML quote needs to be placed where a variable and string constant are concatenated, a triple quote is necessary (intCount &

"""">Hello World</A>").

The example line above will produce the following HTML within the browser (assuming that intCount equals 3).

```
<A HREF = "Example1.asp?HelloCount=3">Hello World</A>
```

Since the hyperlink ASP file here refers to itself, the query string will increase by one the next time it is clicked for each individual user.

The other lines of example code report the number of sequential times this hyperlink was clicked and sends the entire HTML code back to the browser with the MyRequest.Write (strVariable) line.

Further Possibilities with Components

The real power of ASP components is made available by the extra functionality of Visual Basic. Using basic object oriented programming techniques further increases the flexibility and power of server-side ActiveX component programming. The components I write typically incorporate entire multi-page sites using database information, reusable multi-user password systems, remote management, and many other functions that are readily available with code written in Visual Basic (visit my web site for examples http://www.dougdean.com).

Another trick I found useful is creating a centralized public method which manages a multiple page site within a component. The functions that construct my ASP pages, typically from database data, include hyperlinks or redirects with query string variables. I generally have one query string named 'Page' that tracks where a web visitor is within a site. Along with a menu system that uses the same query string system, I can determine where the user wants to go.

My manager method is generally the only public method within my management class. It determines which page to process by reading the MyRequest.QueryString("Page") and then calls the appropriate function using a Select Case statement. The Case statement calls a function that returns the constructed HTML code which the manager Sub then sends to the server.

This modular approach allows me to replace functions and/or classes with optimized code down the line. Reusable classes can be developed which encapsulate functionality for returning HTML code. Setting the class properties can be accomplished within the called functions or within a Case section itself if I'm using classes instead of functions. Here is a simple example of the manager method technique.

```
Public Sub Manager()

     Dim Page As Integer
     Dim strtemp As String

     Select Case CInt(MyRequest.QueryString("Page"))

        Case 1
             strtemp = FrameSet
        Case 2
             strtemp = LeftSideMenu
        Case 3
             strtemp = RightSideWindow
```

```
            Case 4
                 strtemp = ReturnDBinfo
            Case 5
                 strtemp = FormInput

         End Select

        MyResponse.Write (strtemp)

    End Sub
```

With this basic structure you can develop complex sites while retaining manageability.

Download

The source presented in this article is available for download.
[Code Source Download]

About the Author

Doug Dean lives a dual existence as an educational psychologist and independent programmer in southern California. His first commercial program was released in 1985 and was written in 8080 assembler language. “Please Understand Me – The Computer Version”, a personality profiling program, continues to sell and is published by Cambridge Software. Last year, “Deluxe Edition Personality Test” was released by Virtual Entertainment and sells nationally in computer and retail outlets. Excerpts from “Deluxe Edition Personality Test” can be viewed at http://www.dougdean.com/learningstyles/index.cfm.

Doug’s current interests involve developing dynamic web sites, particularly with server-side ActiveX components. He is the author of a number of components -- all with remote managing capabilities and multi-users password protected systems. EZsite Calendar, EZsite WebNotes, and EZsite UpLoad are all available from his web site at http://www.dougdean.com.

Back to article